Karlsruher Institut für Technologie

Übung 8 – Algorithmen II

Tobias Heuer, Sebastian Lamm – *tobias.heuer@kit.edu, lamm@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS19.php`

Institut für Theoretische Informatik - Algorithmik II

Themenübersicht

- Geometrische Datenstrukturen
  - Übersicht
  - Interval Tree
  - Wavelet Tree
- Sweepline
  - Allgemeines Prinzip
  - Einführung: Berechnung einer Skyline
  - Linienschnitt-Algorithmus
- Konvexe Hülle
  - Punktorientierung
  - Graham-Scan Algorithmus

Geometrische Algorithmen

- geomtetrische Varianten bekannter Probleme
  - Spezialfälle oft einfacher
  - allgemeines TSP: nicht approximierbar (wenn $P \neq NP$)
  - metric TSP: 1.5-Approximation
  - euclidean TSP: ϵ-Approximation
    - Laufzeit in $O(n(\log n)^{O(\frac{1}{\epsilon}\sqrt{d})^{d-1}})$
- geometrisch motivierte Probleme
  - Punktlokalisierung
  - Bewegungsplanung (Robotik)
  - Sichtbarkeitsgraphen/Prüfung
  - Streckenschnitt
  - ...

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree (1-dim)
  - Quad Tree (2-dim)
  - k-d-Tree (n-dim)
  - Wavelet-Tree (2-dim)
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm

Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

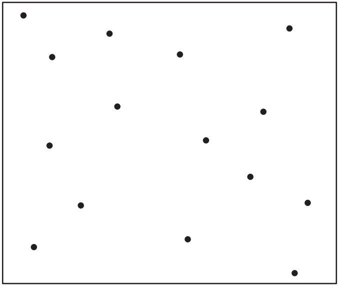
Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree (1-dim)
  - Quad Tree (2-dim)
  - k-d-Tree (n-dim)
  - Wavelet-Tree (2-dim)
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm

Strukturierter Zugriff

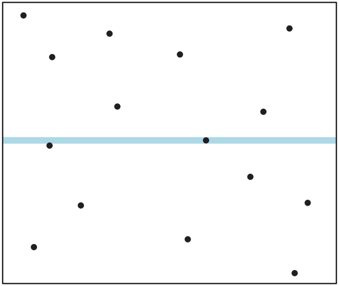
- Sweepline
  - sortiert
  - topologisch sortiert



Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree (1-dim)
  - Quad Tree (2-dim)
  - k-d-Tree (n-dim)
  - Wavelet-Tree (2-dim)
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



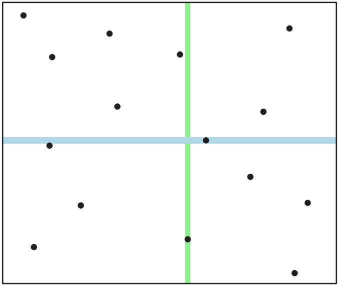
Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree (1-dim)
  - Quad Tree (2-dim)
  - k-d-Tree (n-dim)
  - Wavelet-Tree (2-dim)
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



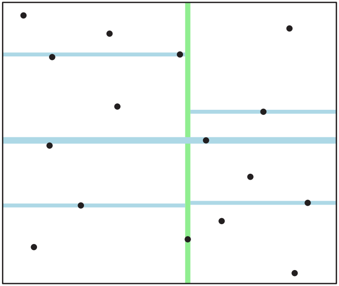
Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree (1-dim)
  - Quad Tree (2-dim)
  - k-d-Tree (n-dim)
  - Wavelet-Tree (2-dim)
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



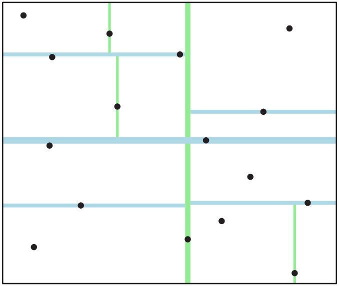
Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree (1-dim)
  - Quad Tree (2-dim)
  - k-d-Tree (n-dim)
  - Wavelet-Tree (2-dim)
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



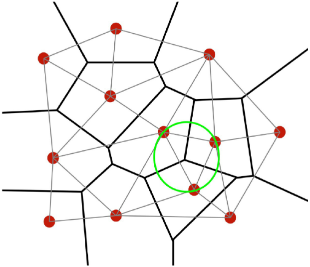
Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree (1-dim)
  - Quad Tree (2-dim)
  - k-d-Tree (n-dim)
  - Wavelet-Tree (2-dim)
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



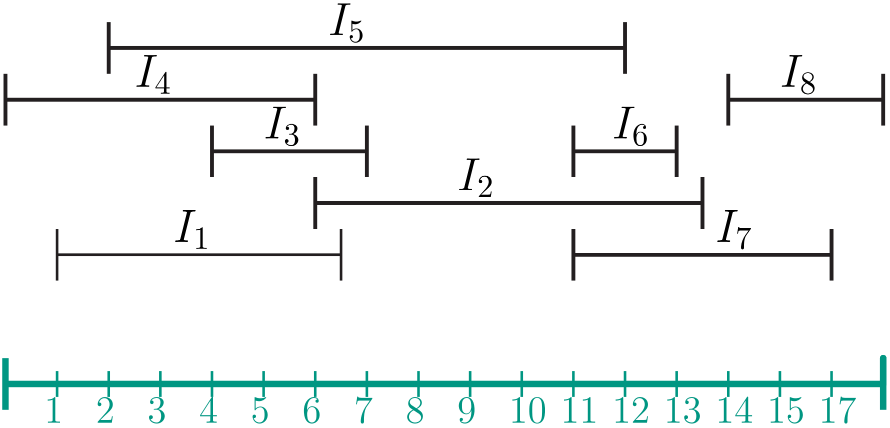
https://i.stack.imgur.com/01H88.png

Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

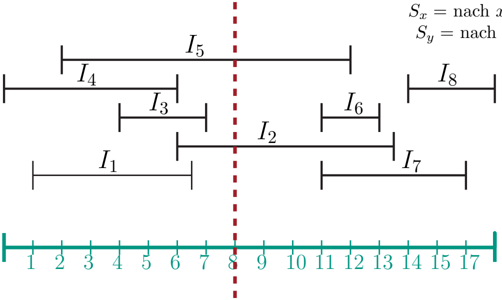
Interval Tree

Konstruktion



Interval Tree

Konstruktion

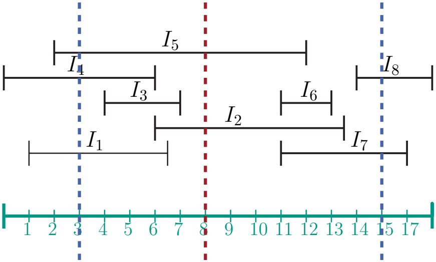


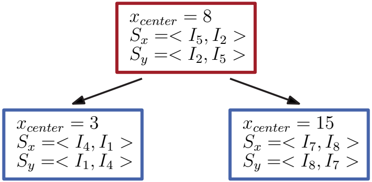
S_x = nach x aufsteigend sortierte Liste
S_y = nach y absteigend sortierte Liste

$$\boxed{\begin{aligned} x_{center} &= 8 \\ S_x &= < I_5, I_2 > \\ S_y &= < I_2, I_5 > \end{aligned}}$$

Interval Tree

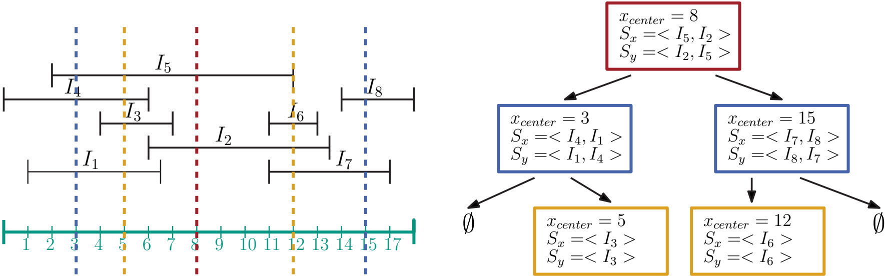
Konstruktion





Interval Tree

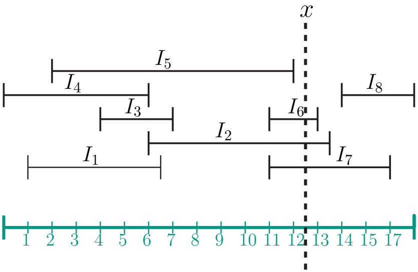
Konstruktion

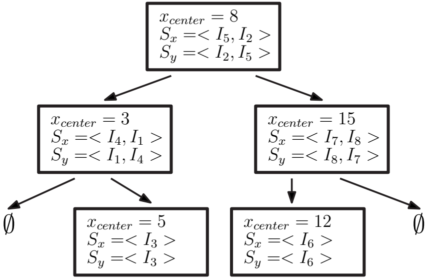


Interval Tree

Schnitt mit Punkt

$I \cap x = <>$

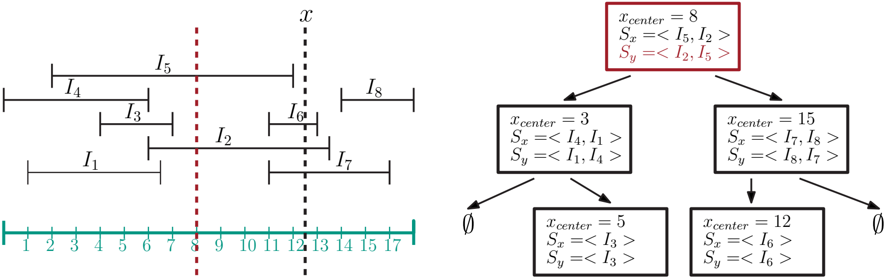




Interval Tree

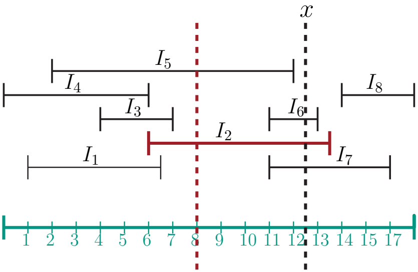
Schnitt mit Punkt

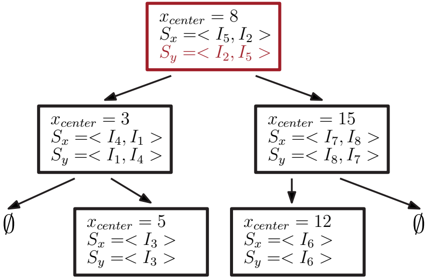
$I \cap x = <>$



Interval Tree
Schnitt mit Punkt

$I \cap x =< I_2 >$

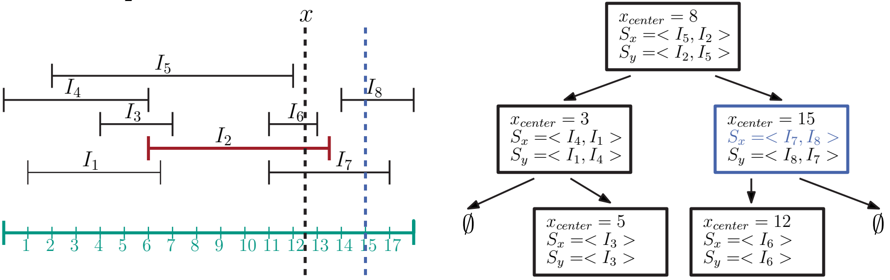




Interval Tree

Schnitt mit Punkt

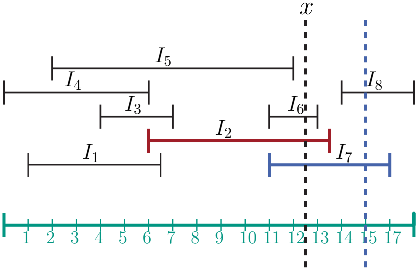
$I \cap x =< I_2 >$

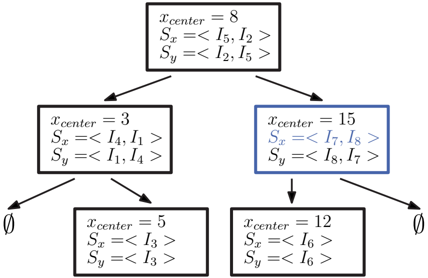


Interval Tree

Schnitt mit Punkt

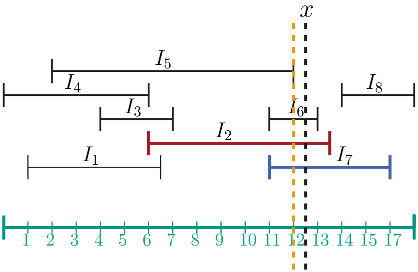
$I \cap x = < I_2, I_7 >$

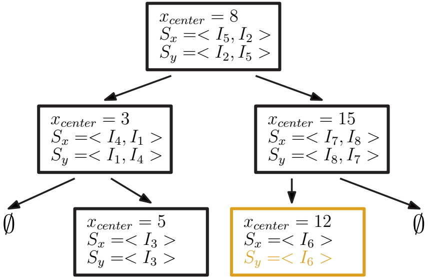




Interval Tree
Schnitt mit Punkt

$I \cap x =< I_2, I_7 >$

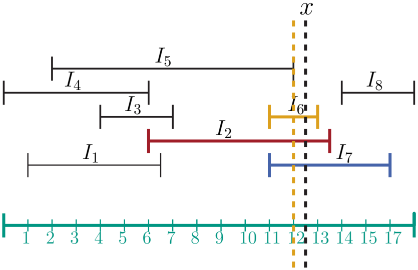


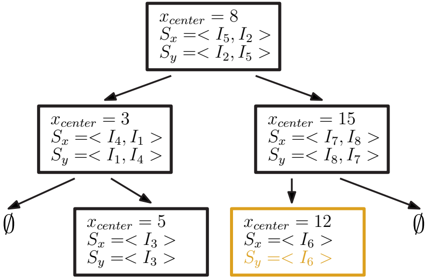


Interval Tree

Schnitt mit Punkt

$I \cap x =< I_2, I_7, I_6 >$

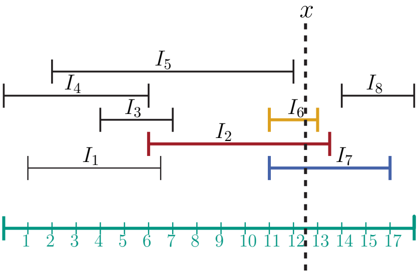


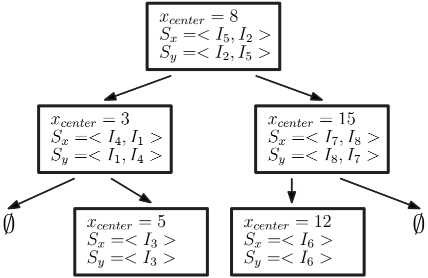


Interval Tree

Schnitt mit Punkt

$I \cap x =< I_2, I_7, I_6 >$





Bitvektoren

Rank und Select

Gegeben ein Bitvektor B der Länge n ($B[i] \in \{0, 1\}$). Wir definieren die folgenden Funktionen auf dem Bitvektor B:

- $rank_1(i, B) =$ Anzahl an 1's in $B[0..i]$
- $select_1(i, B) =$ Position der i-ten 1 in B

$\Rightarrow$ *rank* und *select* können mit $o(n)$ Bits zusätzlichen Speicherplatz und $O(1)$ Laufzeit implementiert werden.

Bitvektoren
Implementierung von Rank

1110 1010 0011 1100 1101 0001 0011 1110 0111 1111 0011

Bitvektoren

Implementierung von Rank

Superblöcke $L = \log^2 n$

1110 1010 0011 1100	1101 0001 0011 1110	0111 1111 0011
0	**9**	**18**

$$Space(rank) = \frac{n}{log^2(n)} \cdot log(n)$$

Bitvektoren
Implementierung von Rank

Superblöcke $L = \log^2 n$

Subblöcke $B = \frac{\log n}{2}$

0	5	0	4	0	7
1110 1010	0011 1100	1101 0001	0011 1110	0111 1111	0011
0		9		18	

$$Space(rank) = \frac{n}{log^2(n)} \cdot log(n) + \frac{2n}{\log(n)} \cdot \log\log^2(n)$$

Bitvektoren

Implementierung von Rank

Superblöcke $L = \log^2 n$

Subblöcke $B = \frac{\log n}{2}$

Lookup-Table für *rank* für jede mögliche Position i und beliebiger Bitvektor der Größe $\frac{\log n}{2}$

0	5	0	4	0	7
1110 1010	0011 1100	1101 0001	0011 1110	0111 1111	0011
0		9		18	

$$Space(rank) = \frac{n}{log^2(n)} \cdot log(n) + \frac{2n}{\log(n)} \cdot \log\log^2(n) + 2^{\frac{\log n}{2}} \cdot \frac{\log n}{2} \cdot \log\log(n)$$

Bitvektoren

Implementierung von Rank

Superblöcke $L = \log^2 n$

Subblöcke $B = \frac{\log n}{2}$

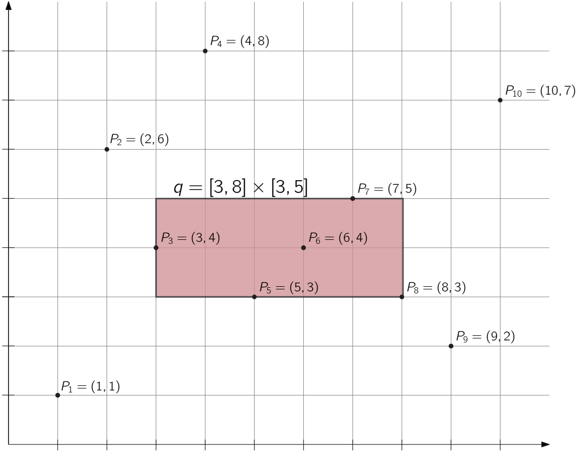
Lookup-Table für *rank* für jede mögliche Position i und beliebiger Bitvektor der Größe $\frac{\log n}{2}$

0	5	0	4	0	7
1110 1010	0011 1100	1101 0001	0011 1110	0111 1111	0011
0		9		18	

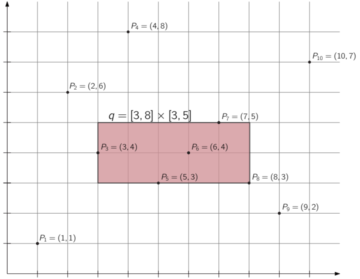
$$Space(rank) = \frac{n}{log^2(n)} \cdot log(n) + \frac{2n}{\log(n)} \cdot \log\log^2(n) + 2^{\frac{\log n}{2}} \cdot \frac{\log n}{2} \cdot \log\log(n)$$

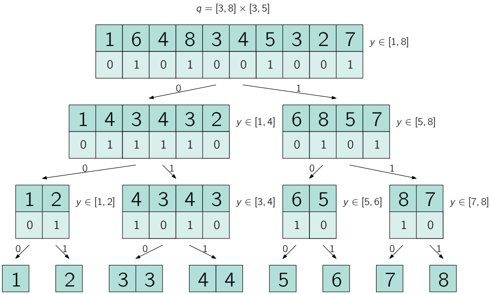
$$= O(\frac{n}{log(n)}) + O(\frac{n}{\log(n)} \cdot \log\log(n)) + O(\sqrt{n} \cdot \log n \cdot \log\log(n)) \in o(n)$$

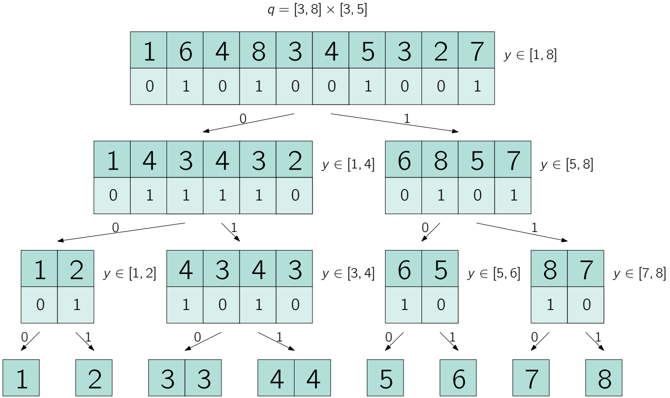
2D Range Queries
Wavelet Tree



2D Range Queries
Wavelet Tree





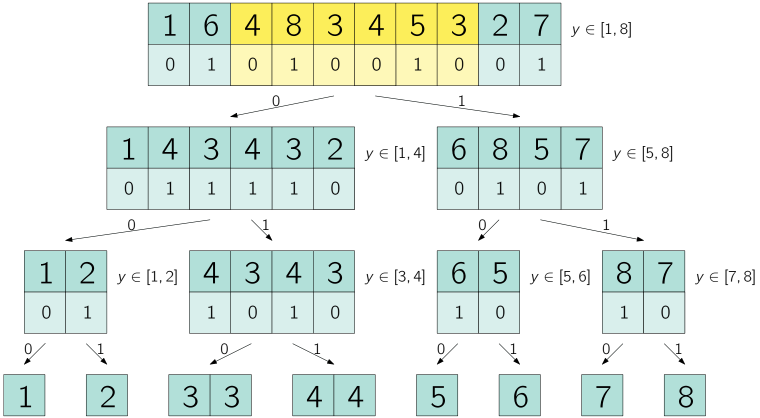


Auf einem $n \times n$ Grid braucht ein Wavelet Tree $n \log n + o(n \log n)$ Bits Speicherplatz

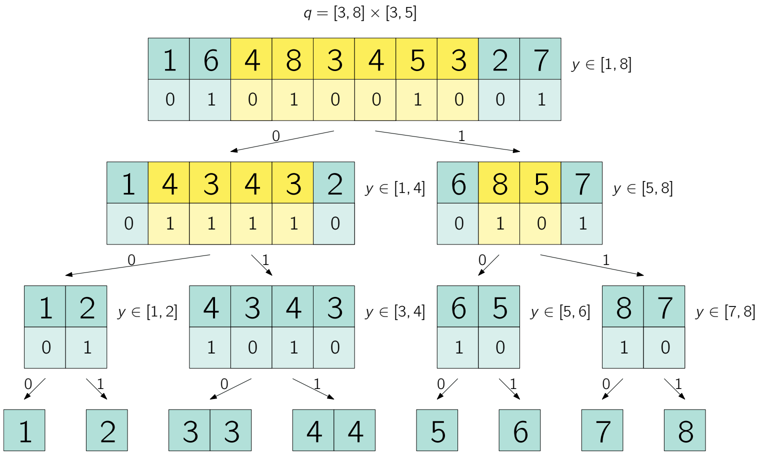
2D Range Queries

Wavelet Tree - Count Operation

$q = [3, 8] \times [3, 5]$

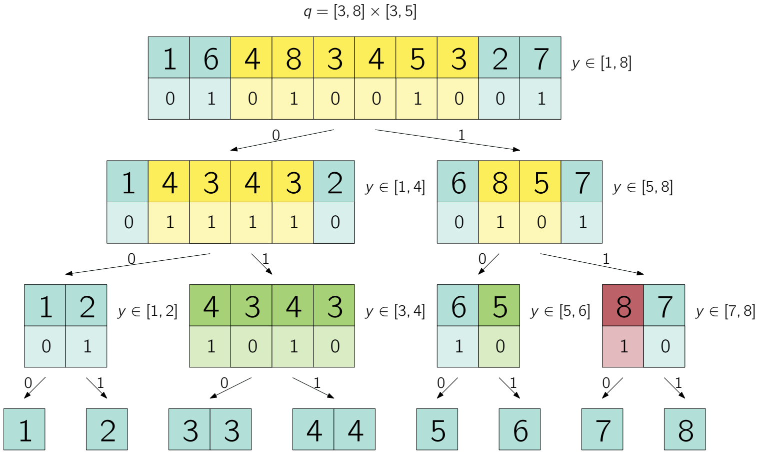


2D Range Queries
Wavelet Tree - Count Operation



2D Range Queries

Wavelet Tree - Count Operation



$\mathcal{O}(\log n)$ Baumknoten werden besucht bevor die Recursion stoppt $\Rightarrow$ count hat Laufzeit $\mathcal{O}(\log n)$

Sweep-Line
Allgemein

Idee
- strukturierte Abarbeitung eines Problems
- nutze Nähe aus
  - geometrisch nahe Objekte beeinflussen sich
  - geometrisch weit entfernte Objekte (nahezu) unabhängig

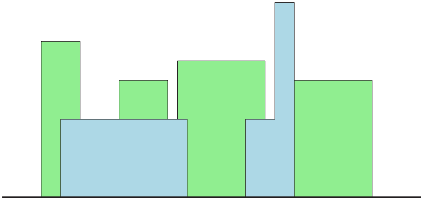
im Allgemeinen
- reduziere n-dim $\rightarrow$ $(n-1)$-dim

Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

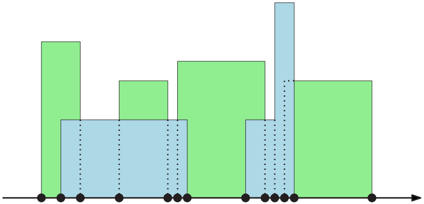


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

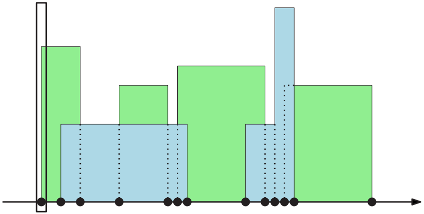


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

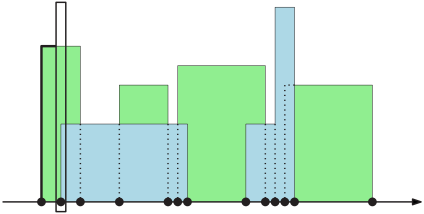


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

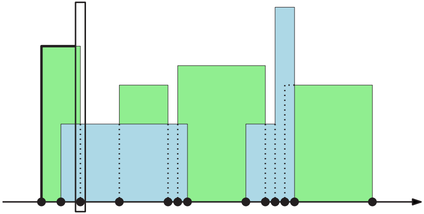


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

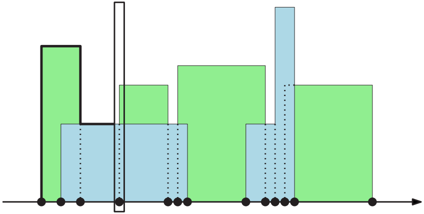


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

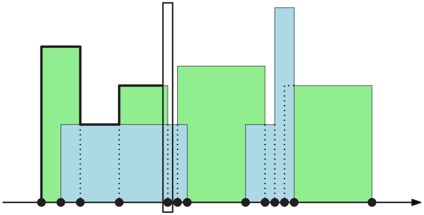


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

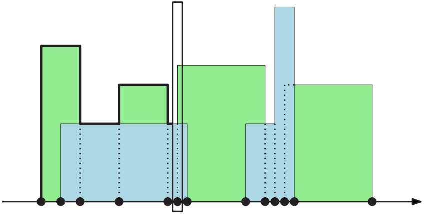


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

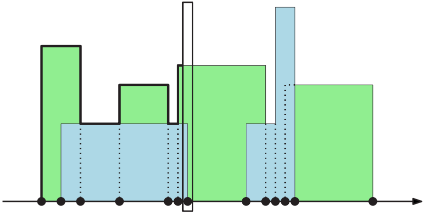


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

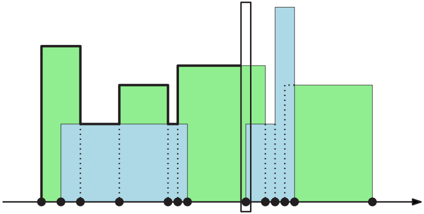


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

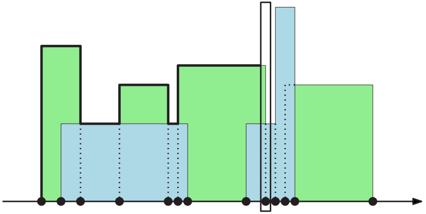


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

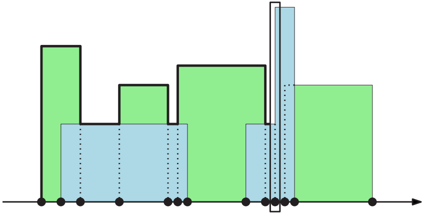
- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)



Sweep-Line
Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

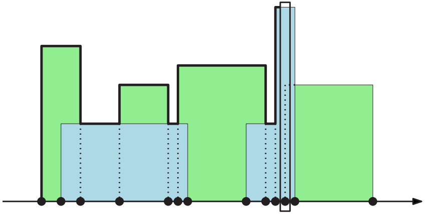


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

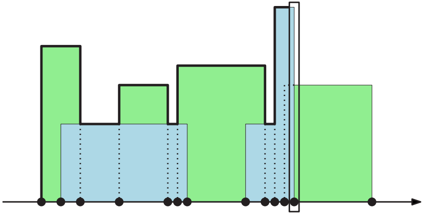


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

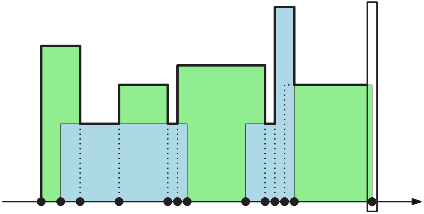


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

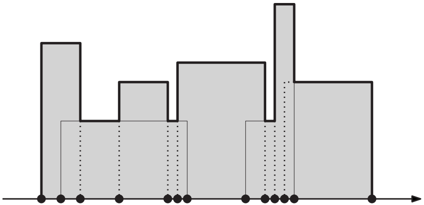


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  (Maximumsbildung)

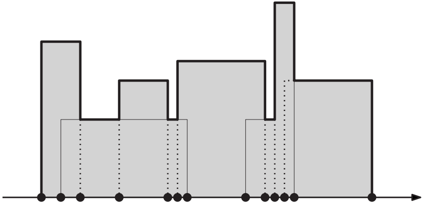


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)



Problem: effiziente Lösung des 1-dimensionalen Problems

- ineffizient: $\mathcal{O}(n^2)$ (vergleiche Linienschnitt)
- Ziel hier: Algorithmus mit $O(n \log n)$ Zeit

One-Dimensional Problem

Sortierte Liste

- Array $\rightarrow \mathcal{O}(n)$ für Einfügen/Löschen
- Linked List $\rightarrow \mathcal{O}(n)$ für Positionsbestimmung

One-Dimensional Problem

Sortierte Liste

- Array $\rightarrow \mathcal{O}(n)$ für Einfügen/Löschen
- Linked List $\rightarrow \mathcal{O}(n)$ für Positionsbestimmung

Lösung: Priority Queue

- alle Operationen in maximal $\mathcal{O}(\log n)$ möglich

Pseudocode (1/2)

Data: List of buildings (begin, height, end)
Result: Skyline coordinates (x, height)
$i \leftarrow 0$;
foreach $(b, h, e) \in L$ **do**
  $L' \leftarrow L' \cup (b, h, \text{“}b\text{“}, i) \cup (e, g, \text{“}e\text{“}, i)$;
  $i \leftarrow i + 1$;
end
sort L' in lexicographical order;
priority queue $q \leftarrow \emptyset$;
...

Pseudocode (2/2)

```
...
while L′ ≠ ∅ do
    actpos ← first(L′).pos;
    while L′ ≠ ∅ and actpos = first(L′).pos do
        (pos,height,label,index) ← popfirst(L′);
        if label = “b“ then
            add (q, (height,index));
        else
            remove (q,index);
        end
    end
    if q ≠ ∅ then
        print actpos,first(q).height;
    else
        print actpos,0;
    end
end
```

Linienschnitt

überlappende Liniensegmente

Events

1. *Start*-Event $e := (y, \text{start}, s = \overline{(x, y)(x', y')})$
2. *End*-Event $e := (y', \text{end}, s = \overline{(x, y)(x', y')})$
3. *Intersection*-Event $e := (y, \text{intersection}, (s_i, s_j))$
   - s_i und s_j sind Linien die sich im Intersection-Event schneiden

Initialisierung

- Speichere *Start*- und *Stop*-Events in einer Priority Queue Q (nach y-Wert sortiert) $(O(n \log n))$

Linienschnitt

überlappende Liniensegmente

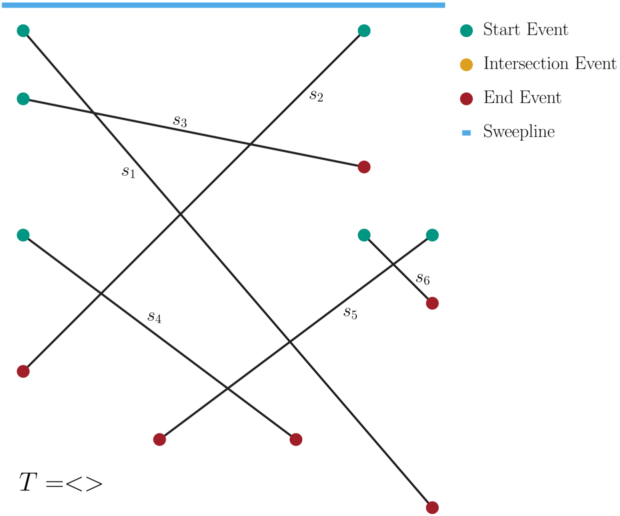
Linearer Scan

1. Entferne nächstes Event aus Priority-Queue Q ($O(\log n)$)
   - 1.1 *Start*-Event: Füge s in eine sortierte Liste T hinzu (nach x-Koordinate) und teste mit *Nachbarn* in T auf Linienschnitt ($O(\log n)$)
   - 1.2 *End*-Event: Lösche s aus sortierter Liste T und teste beide vorherigen *Nachbarn* in T auf Linienschnitt ($O(\log n)$)
   - 1.3 *Intersection*-Event: Vertausche s_i und s_j in sortierter Liste T und teste s_i mit seinem neuen rechten und s_j mit seinem neuen linken *Nachbarn* in T auf Linienschnitt ($O(\log n)$)
   - 1.4 Falls ein Schnitttest zweier Linien einen Schnittpunkt ergibt, füge *Intersection*-Event zu Q hinzu ($O(\log n)$)

$\Rightarrow$ Laufzeit $O((n+k)\log n)$ wobei k Anzahl an Schnittpunkten

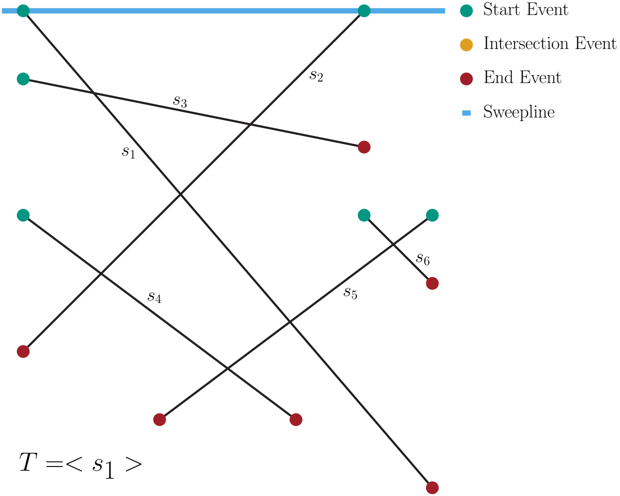
Linienschnitt

Illustration Sweepline-Algorithmus



Linienschnitt

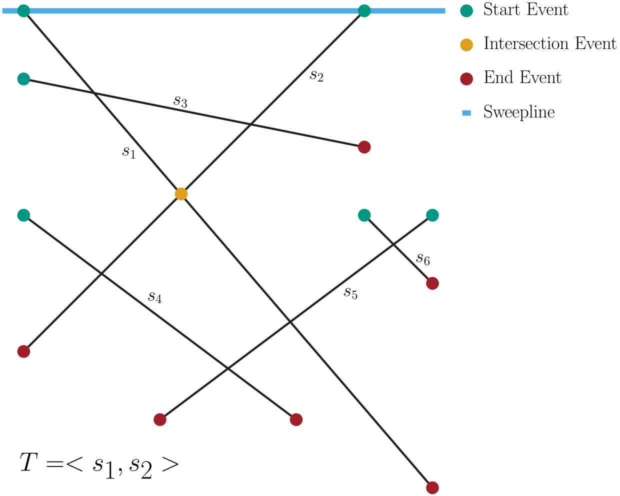
Illustration Sweepline-Algorithmus



$T =< s_1 >$

Linienschnitt

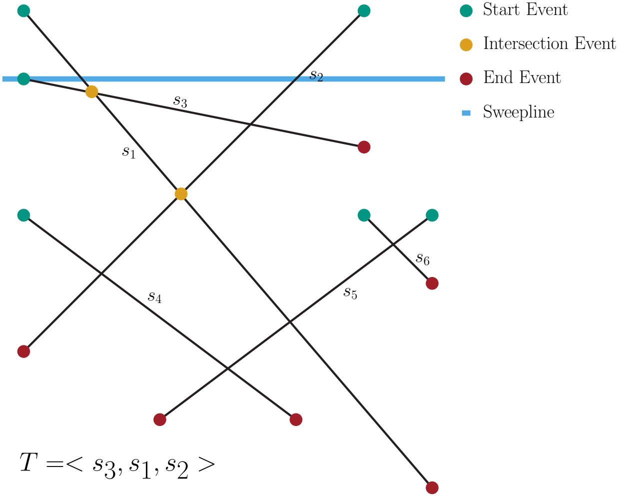
Illustration Sweepline-Algorithmus



$T =< s_1, s_2 >$

Linienschnitt

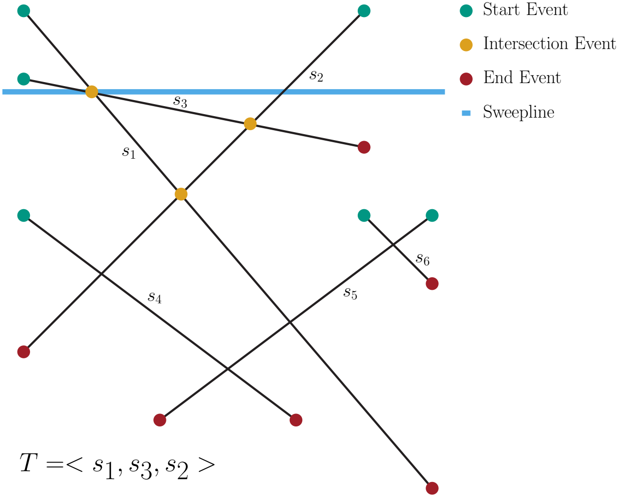
Illustration Sweepline-Algorithmus



$T = < s_3, s_1, s_2 >$

Linienschnitt

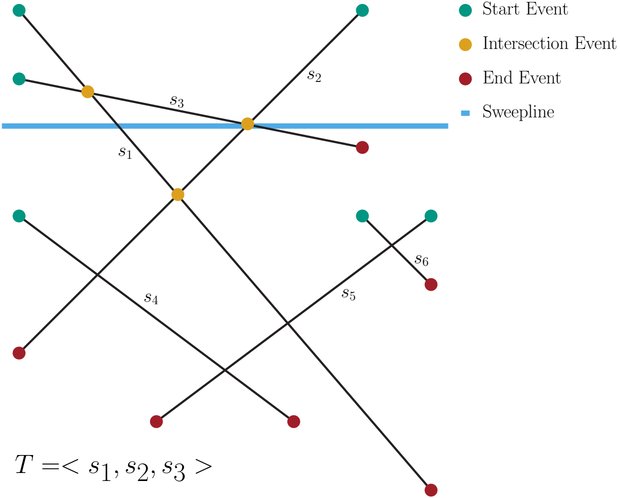
Illustration Sweepline-Algorithmus



$T =< s_1, s_3, s_2 >$

Linienschnitt

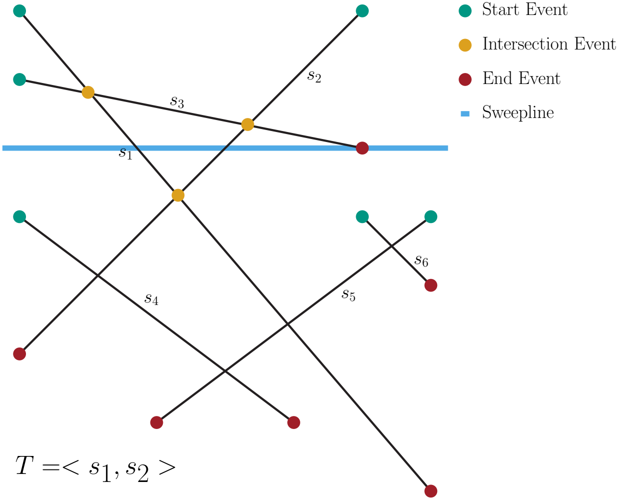
Illustration Sweepline-Algorithmus



$T =< s_1, s_2, s_3 >$

Linienschnitt

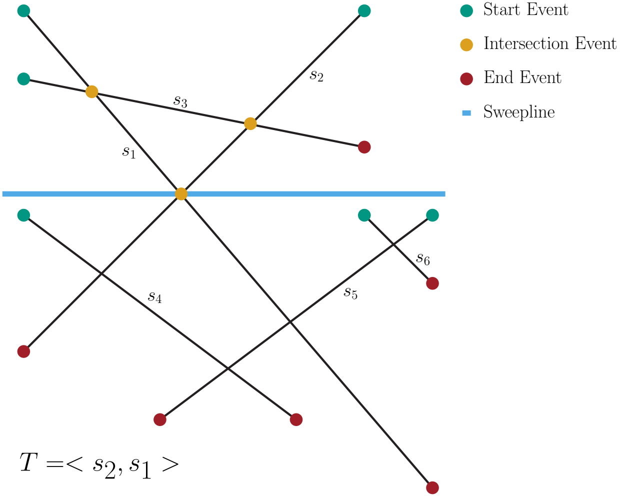
Illustration Sweepline-Algorithmus



$T =< s_1, s_2 >$

Linienschnitt

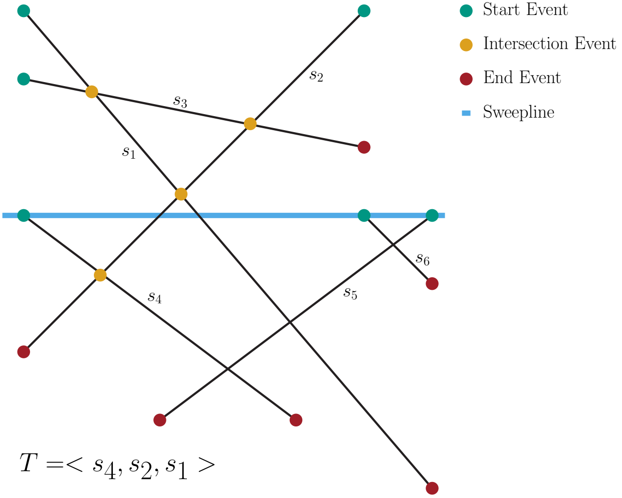
Illustration Sweepline-Algorithmus



$T =< s_2, s_1 >$

Linienschnitt

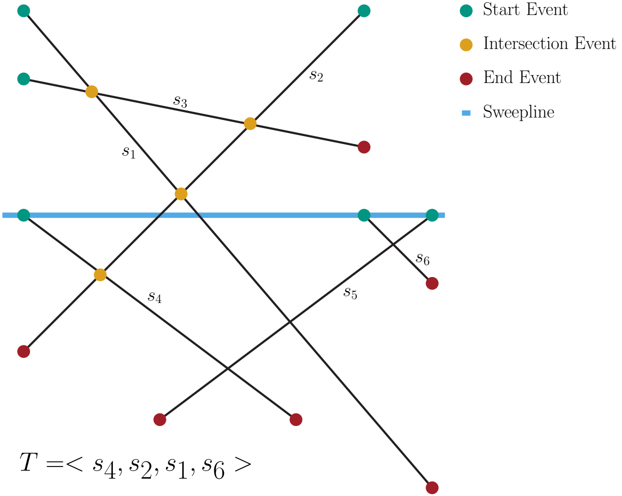
Illustration Sweepline-Algorithmus



$T =< s_4, s_2, s_1 >$

Linienschnitt

Illustration Sweepline-Algorithmus

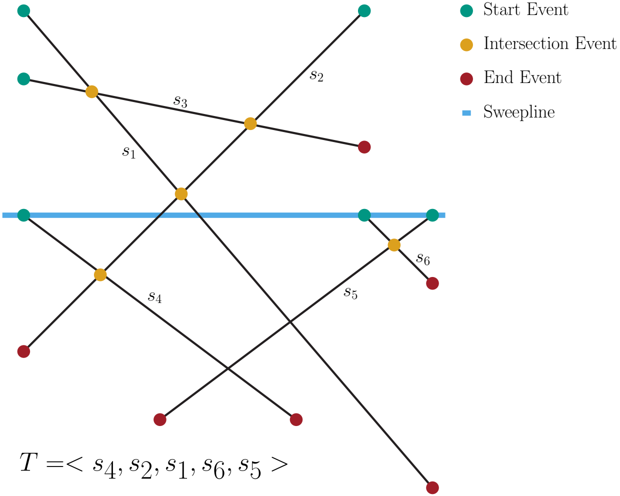


$T =< s_4, s_2, s_1, s_6 >$

Linienschnitt

Illustration Sweepline-Algorithmus

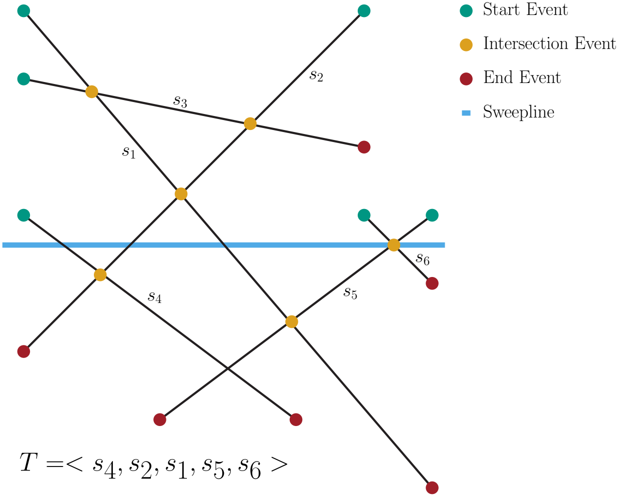
Karlsruher Institut für Technologie



$T = < s_4, s_2, s_1, s_6, s_5 >$

Linienschnitt

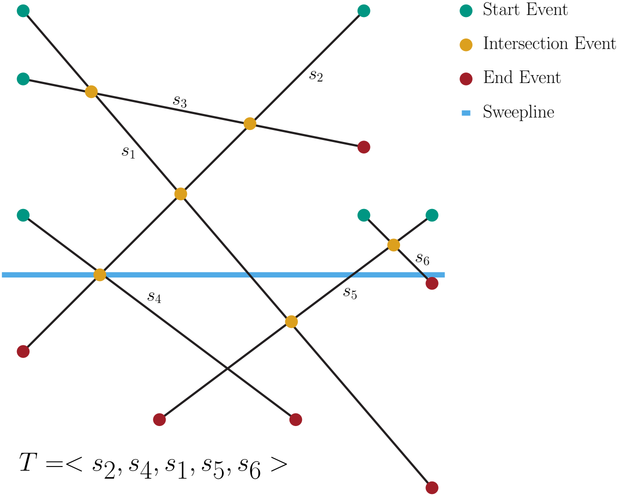
Illustration Sweepline-Algorithmus



$T =< s_4, s_2, s_1, s_5, s_6 >$

Linienschnitt

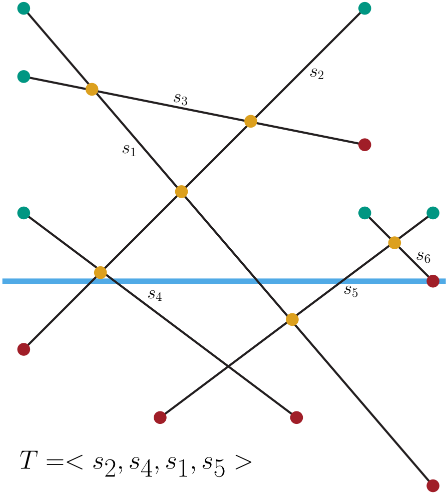
Illustration Sweepline-Algorithmus



$$T =< s_2, s_4, s_1, s_5, s_6 >$$

Linienschnitt

Illustration Sweepline-Algorithmus

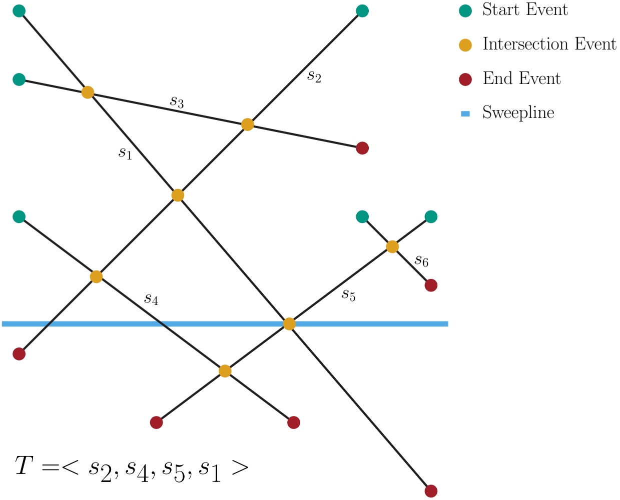


- Start Event
- Intersection Event
- End Event
- Sweepline

$T = < s_2, s_4, s_1, s_5 >$

Linienschnitt

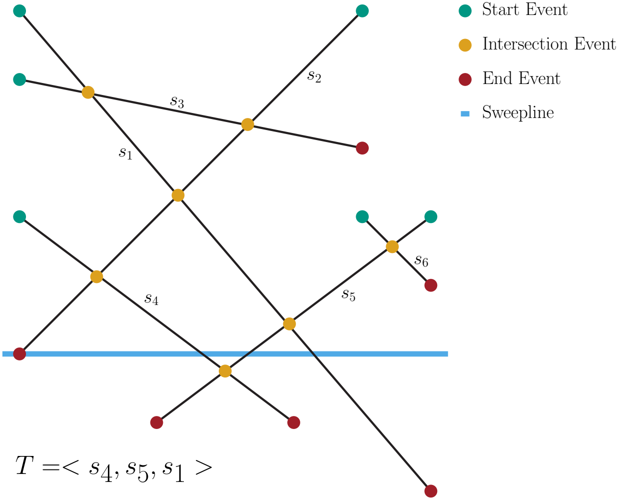
Illustration Sweepline-Algorithmus



$$T = < s_2, s_4, s_5, s_1 >$$

Linienschnitt

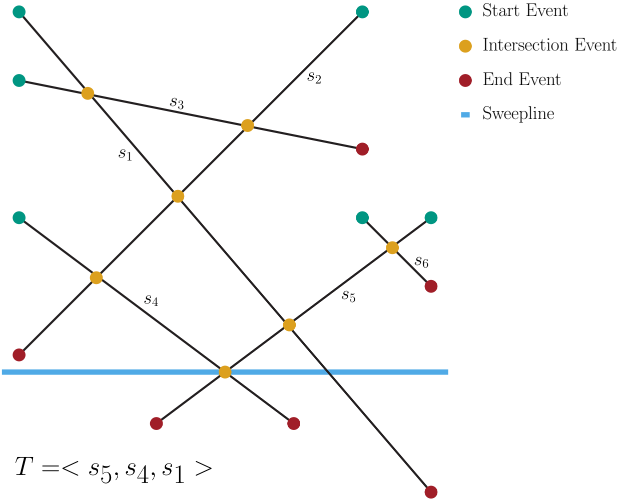
Illustration Sweepline-Algorithmus



$T = < s_4, s_5, s_1 >$

Linienschnitt

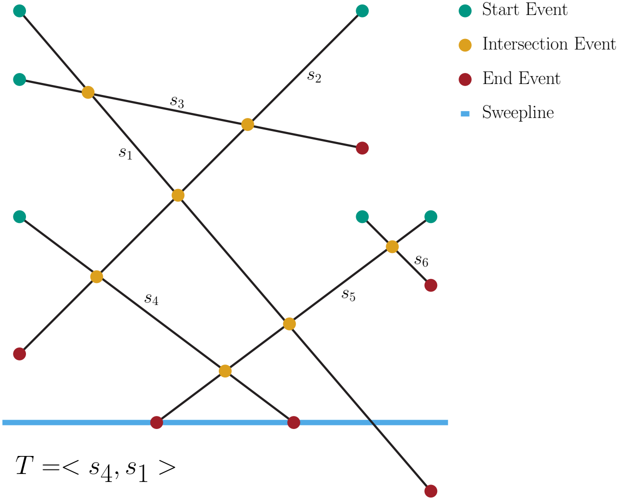
Illustration Sweepline-Algorithmus



$$T = < s_5, s_4, s_1 >$$

Linienschnitt

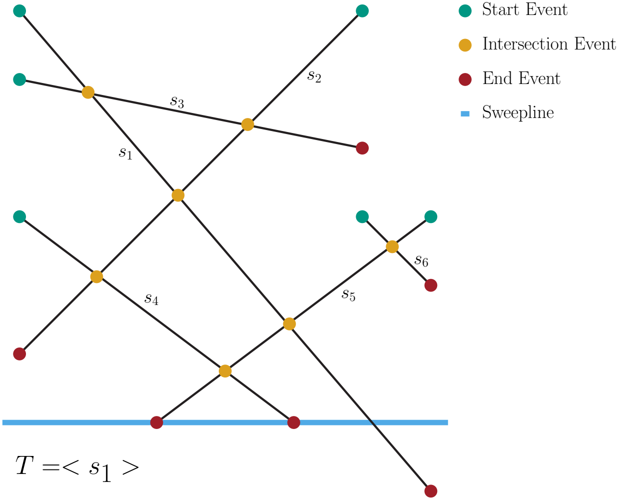
Illustration Sweepline-Algorithmus



$T = < s_4, s_1 >$

Linienschnitt

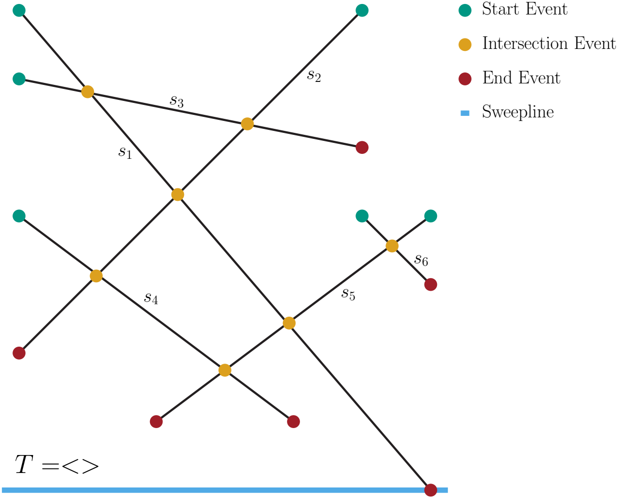
Illustration Sweepline-Algorithmus



$T = < s_1 >$

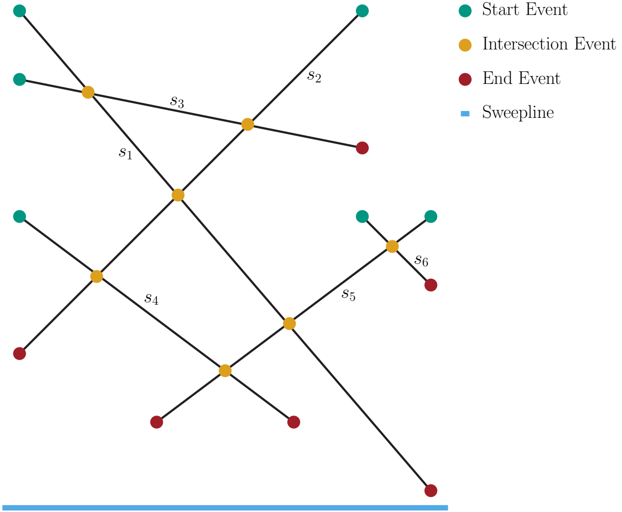
Linienschnitt

Illustration Sweepline-Algorithmus



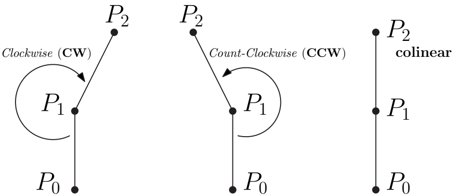
Linienschnitt

Illustration Sweepline-Algorithmus

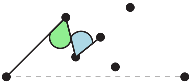


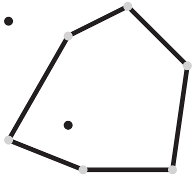
Punktorientierung

- Gegeben drei Punkte P_0, P_1, P_2 $\Rightarrow$ bestimme Orientierung
- Sei $\overrightarrow{a} = \overrightarrow{P_0P_2}$ und $\overrightarrow{b} = \overrightarrow{P_0P_1}$
- $CCW(P_0, P_1, P_2) = a_x \cdot b_y - a_y \cdot b_x$
- $CCW(P_0, P_1, P_2) < 0 \Rightarrow$ **CCW**
- $CCW(P_0, P_1, P_2) = 0 \Rightarrow$ **Colinear**
- $CCW(P_0, P_1, P_2) > 0 \Rightarrow$ **CW**



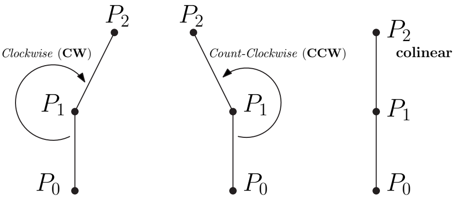
- Unterproblem von Algorithmen
  - Graham Scan
  - Test auf Enthaltensein
    (Punkt in konvexem Polygon)



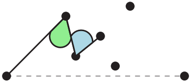


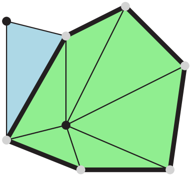
Punktorientierung

- Gegeben drei Punkte P_0, P_1, P_2 $\Rightarrow$ bestimme Orientierung
- Sei $\overrightarrow{a} = \overrightarrow{P_0 P_2}$ und $\overrightarrow{b} = \overrightarrow{P_0 P_1}$
- $CCW(P_0, P_1, P_2) = a_x \cdot b_y - a_y \cdot b_x$
- $CCW(P_0, P_1, P_2) < 0 \Rightarrow$ **CCW**
- $CCW(P_0, P_1, P_2) = 0 \Rightarrow$ **Colinear**
- $CCW(P_0, P_1, P_2) > 0 \Rightarrow$ **CW**



- Unterproblem von Algorithmen
  - Graham Scan
  - Test auf Enthaltensein
    (Punkt in konvexem Polygon)

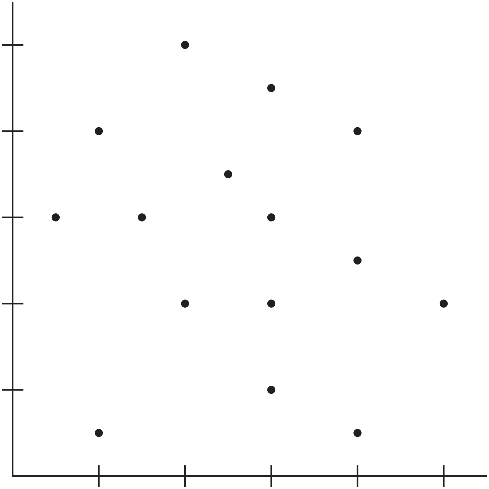




Konvexe Hülle

Graham-Scan

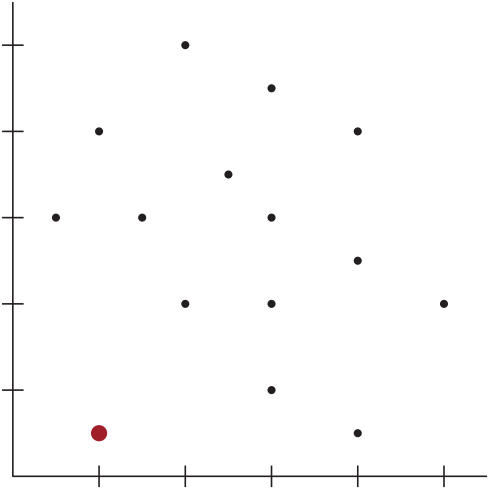
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

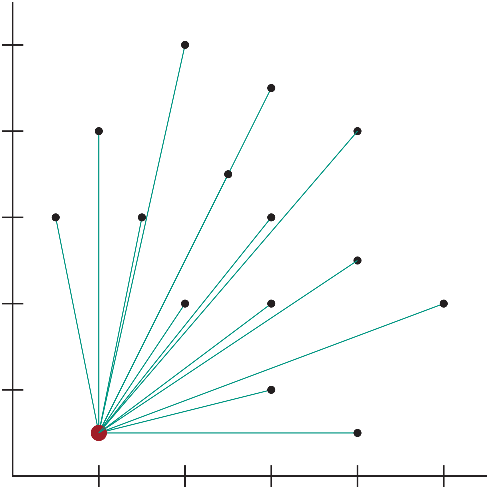
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

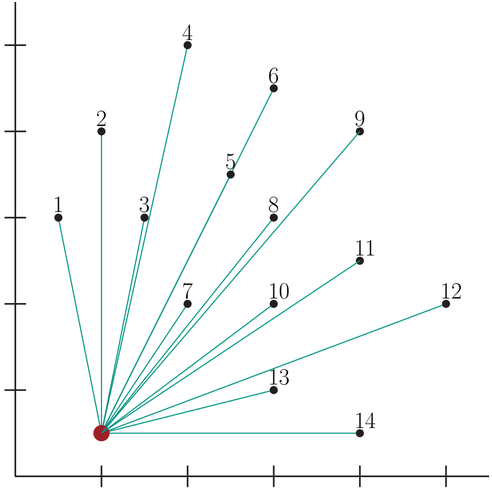
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

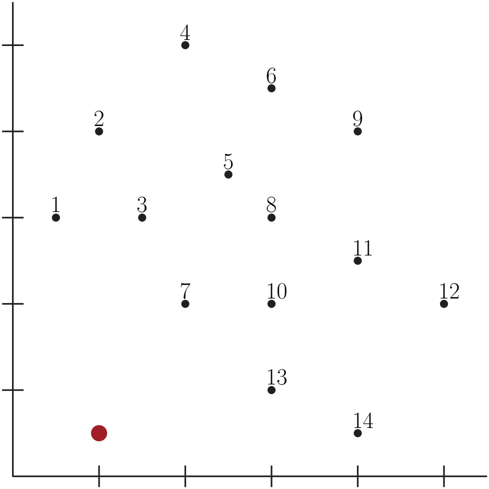
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

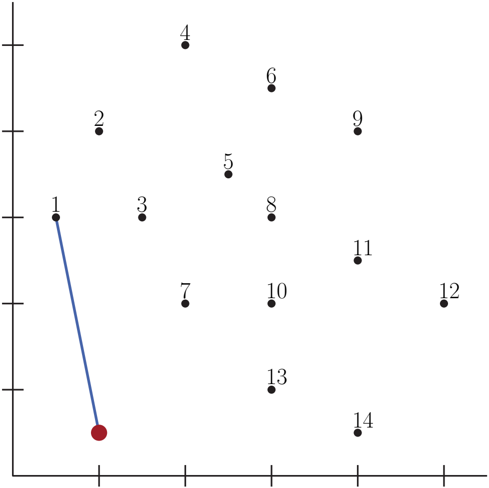
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

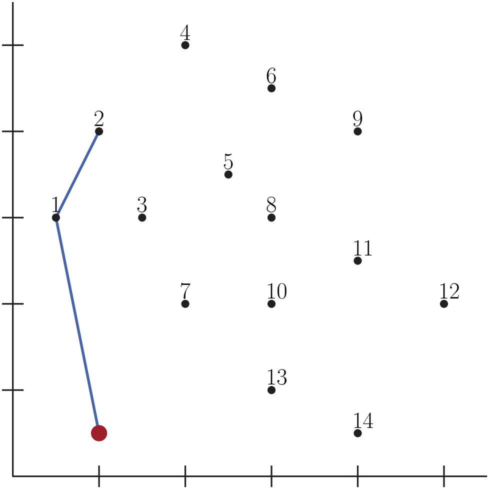
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

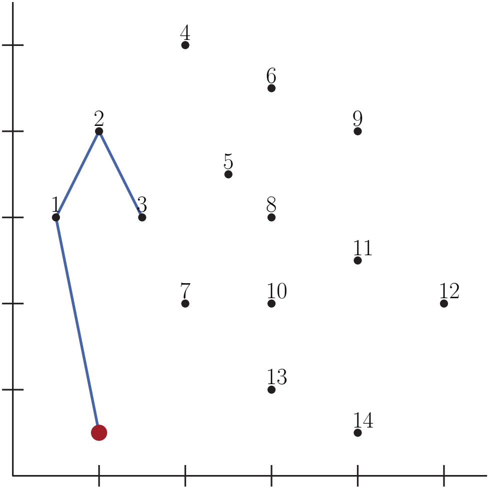
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

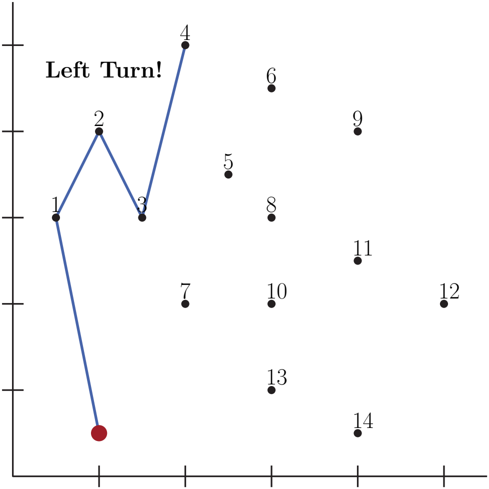
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

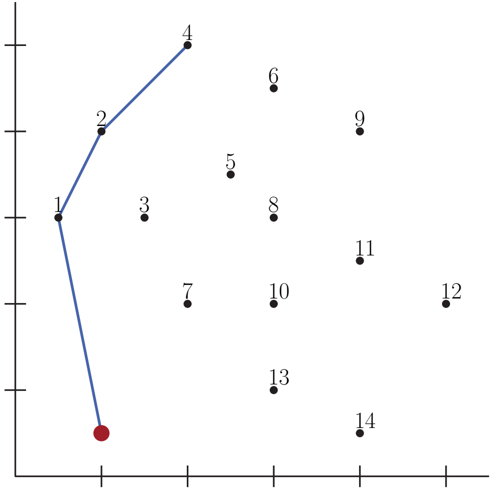
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

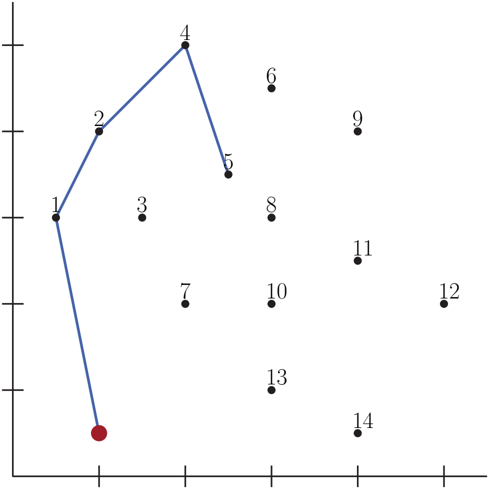
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

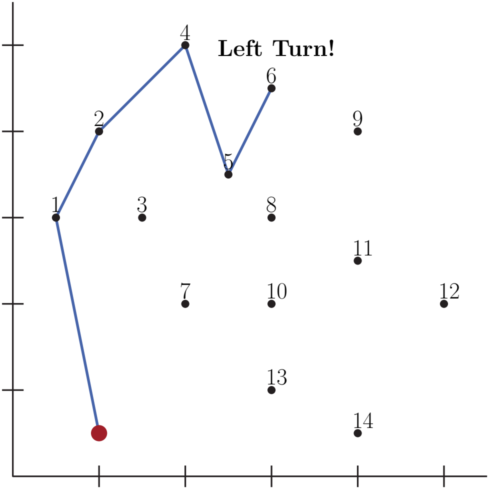
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

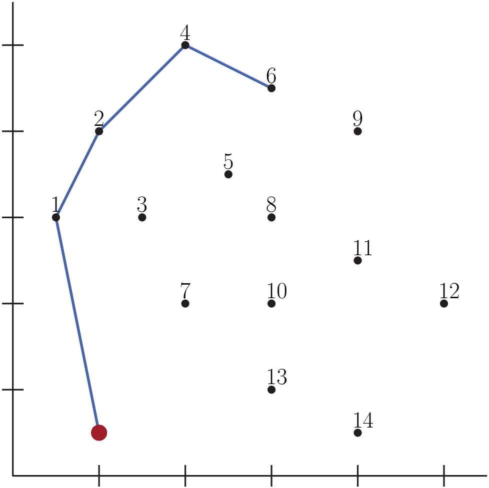
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

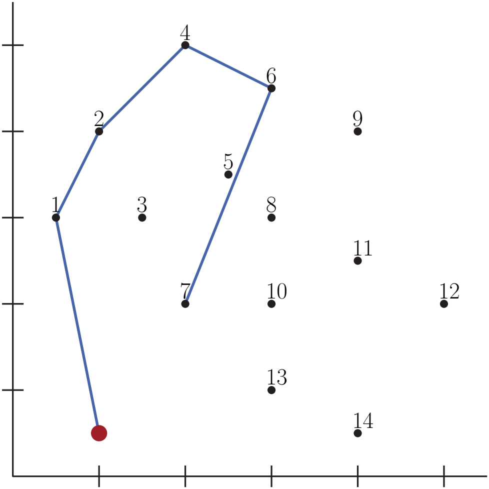
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

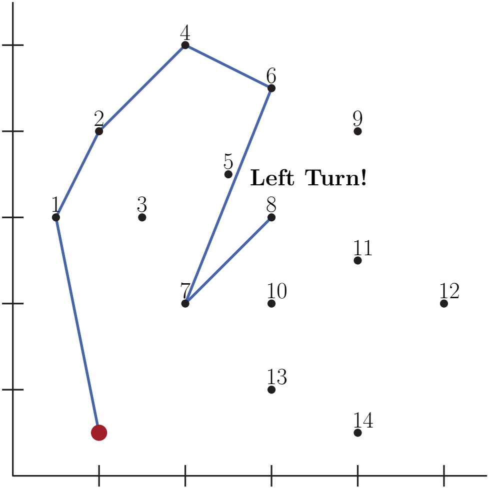
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

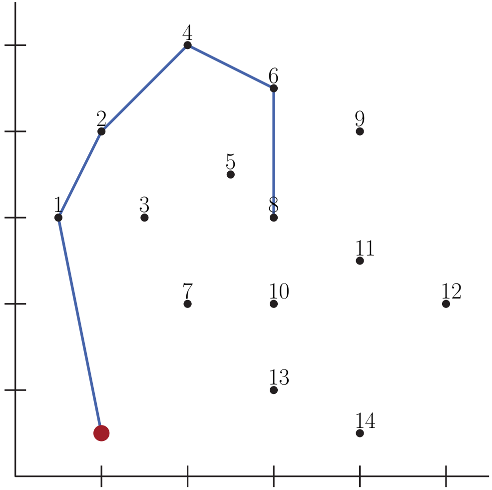
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

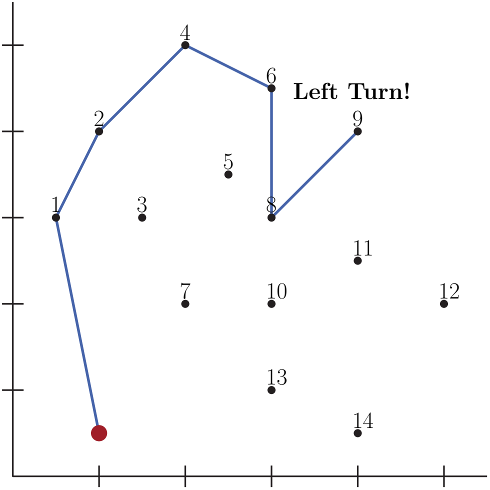
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

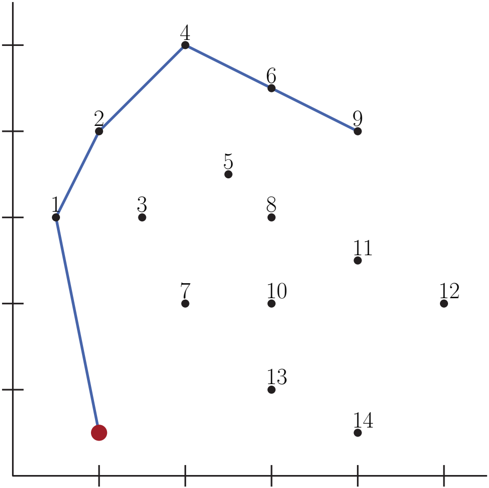
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

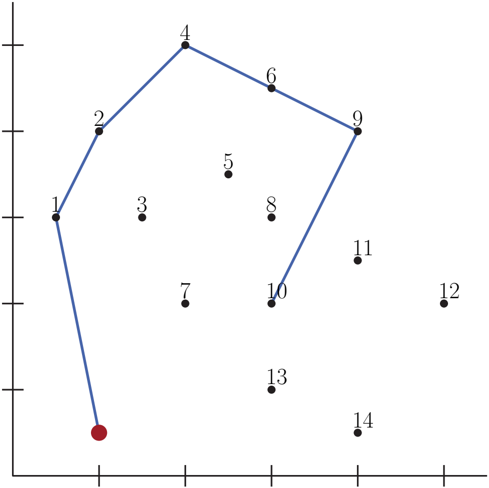
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

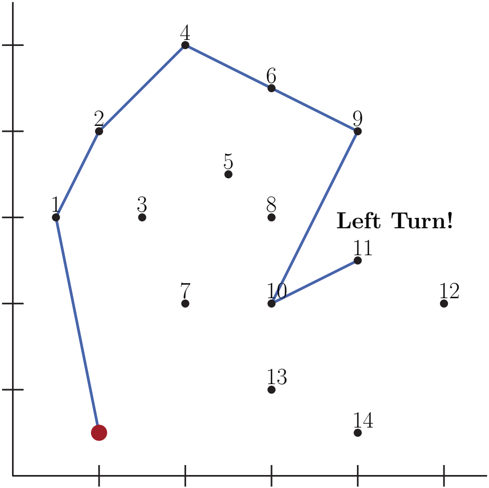
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

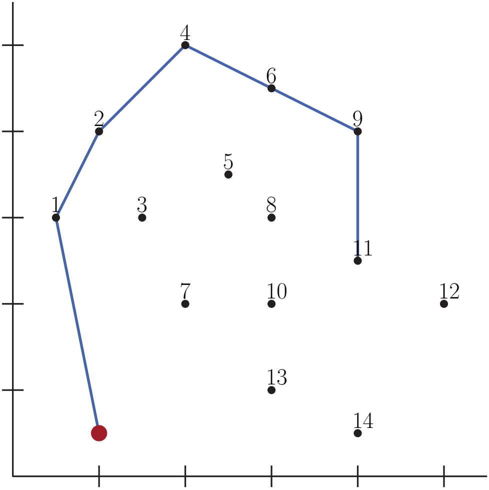
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

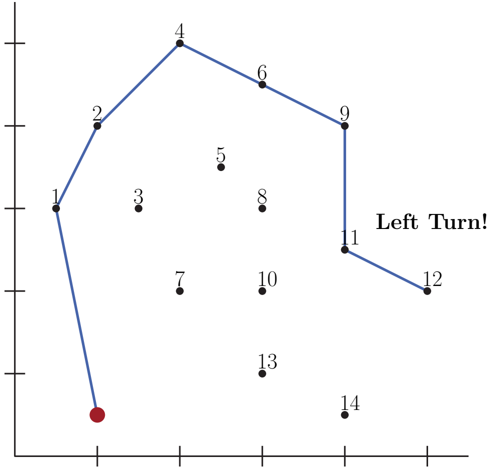
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

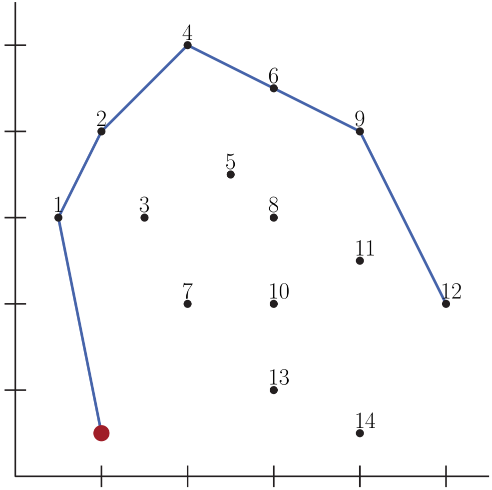
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i $(i > 2)$ und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

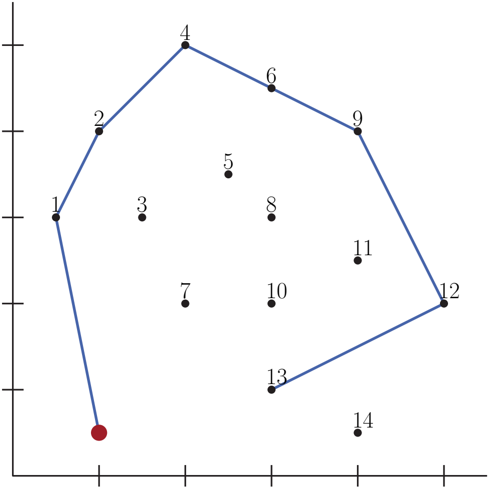
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i $(i > 2)$ und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

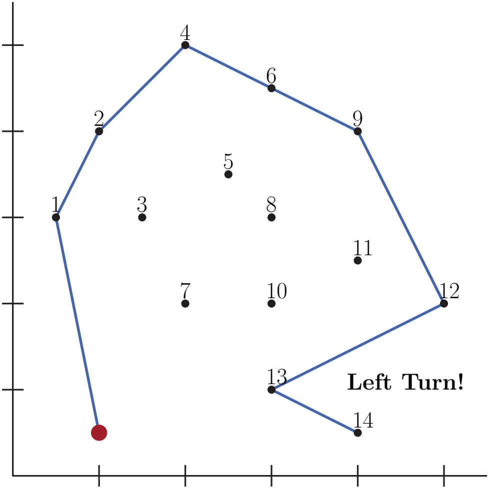
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

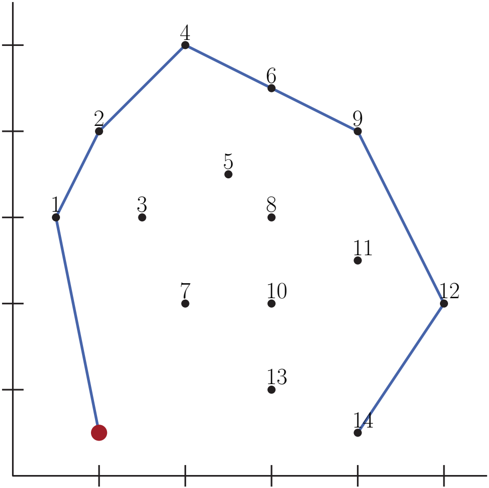
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

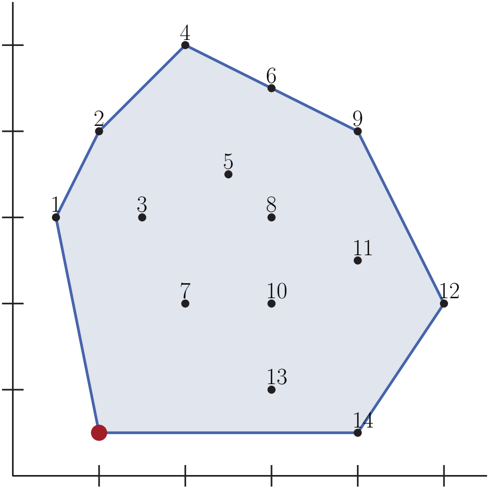
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Ende!

Feierabend!